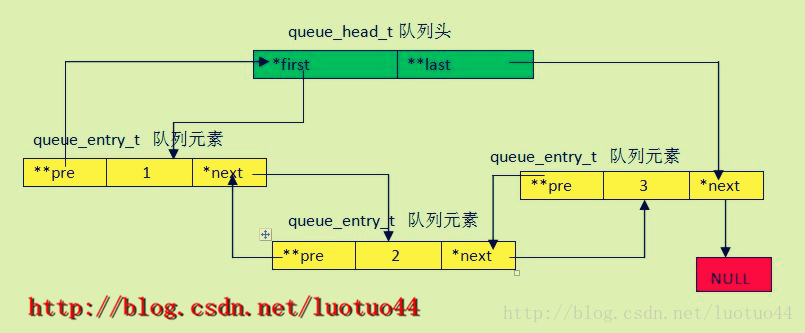
queue.h位于compat/sys目录下。该文件里面定义了多个数据结构，其中TAILQ\_QUEUE是使得最广泛的。

**队列结构体：**

        TAILQ\_QUEUE由下面两个结构体一起配合工作。

1. #define TAILQ\_HEAD(name, type)   //队列头 ，用于管理队列中的元素
2. **struct** name {                               \
3. **struct** type \*tqh\_first; /\* first element \*/         \
4. **struct** type \*\*tqh\_last; /\* addr of last next element \*/     \
5. }
7. //和前面的TAILQ\_HEAD不同，这里的结构体并没有name.即没有结构体名。
8. //所以该结构体只能作为一个匿名结构体。所以，它一般都是另外一个结构体
9. //或者共用体的成员
10. #define TAILQ\_ENTRY(type)      //通过这两个指针将具体的元素串联成队列
11. **struct** {                                \
12. **struct** type \*tqe\_next;  /\* next element \*/          \
13. **struct** type \*\*tqe\_prev; /\* address of previous next element \*/  \
14. }

        由这两个结构体配合构造出来的队列一般如下图所示：



        图中，一级指针指向的是queue\_entry\_t这个结构体，即存储queue\_entry\_t这个结构体的地址值。二级指针存储的是一级地址变量的地址值。所以二级指针指向的是图中的一级指针，而非结构体。图中的1,2, 3为队列元素保存的一些值。

**队列操作宏函数以及使用例子：**

        除了这两个结构体，在queue.h文件中，还为TAILQ\_QUEUE定义了一系列的访问和操作函数。很不幸，它们是一些宏定义。这里就简单贴几个函数（准确来说，不是函数）的代码。

1. #define TAILQ\_FIRST(head) ((head)->tqh\_first)
2. #define TAILQ\_END(head) NULL
3. #define TAILQ\_NEXT(elm, field) ((elm)->field.tqe\_next)
4. #define TAILQ\_LAST(head, headname) \*(((struct headname \*)((head)->tqh\_last))->tqh\_last)) \
5. #define TAILQ\_PREV(elm, headname, field) \
6. (\*(((struct headname \*)((elm)->field.tqe\_prev))->tqh\_last))
7. #define TAILQ\_EMPTY(head) \
8. (TAILQ\_FIRST(head) == TAILQ\_END(head))
9. #define TAILQ\_FOREACH(var, head, field) \
10. for((var) = TAILQ\_FIRST(head); \
11. (var) != TAILQ\_END(head); \
12. (var) = TAILQ\_NEXT(var, field))
13. #define TAILQ\_FOREACH\_REVERSE(var, head, headname, field) \
14. for((var) = TAILQ\_LAST(head, headname); \
15. (var) != TAILQ\_END(head); \
16. (var) = TAILQ\_PREV(var, headname, field))
17. /\*
18. \* Tail queue 相应的功能，在以后的使用中，可以不必了解其具体实现的细节
19. \*/
20. TAILQ\_INIT(head)
21. TAILQ\_INSERT\_HEAD(head, elm, field)
22. TAILQ\_INSERT\_TAIL(head, elm, field)
23. TAILQ\_INSERT\_AFTER(head, listelm, elm, field)
24. TAILQ\_INSERT\_BEFORE(listelm, elm, field)
25. TAILQ\_REMOVE(head, elm, field)
26. TAILQ\_REPLACE(head, elm, elm2, field)

        这些宏是很难看的，也没必要直接去看这些宏。。在struct event{}结构体中，用到了Tail queue，以此为例对其用法进行简要分析

1. //队列中的元素结构体。它有一个值，并且有前向指针和后向指针
2. //通过前后指针，把队列中的节点(元素)连接起来
3. **struct** event\_t  //该结构体 中有很多变量，同时也有多个TAILQ，以其中的IO事件为例
4. {
5. int value; //将其它所有成员当做一个变量
6. LIST\_ENTRY (event\_t) ev\_io\_next;
7. } ev\_;
8. /\*将其中的宏展开，即可得到如下结构
9. struct event\_t
10. {
11. int val;
12. struct {    //通过next,prev指针将所有事件连接成队列
13. struct event\_t \*tqe\_next;            \
14. struct event\_t \*\*tqe\_prev;
15. }
16. } ev\_;
17. \*/
18. //定义一个队列头，结构体名为queue\_head\_t，成员变量类型为event\_t
19. //它有两个指针，tqh\_first,tqh\_last,分别指向队列的头和尾
20. TAILQ\_HEAD(queue\_head\_t,event\_t);
22. int main(){
23. struct queue\_head\_t queue\_head;
24. struct event\_t \*p,\*q,\*s,\*t;
25. TAILQ\_INIT(&queue\_head);
26. assert(queue\_head.tqh\_first==NULL);
27. for(int i=0;i<3;++i){
28. p=new event\_t();
29. p->val=i;
30. TAILQ\_INSERT\_TAIL(&queue\_head,p,ev\_next); //p在队列最尾部，三个元素分别为0,1,2
31. }
32. q=TAILQ\_FIRST(&queue\_head); //队列首元素为0
33. assert(TAILQ\_END(&queue\_head)==NULL);
34. assert(q->val==0);
35. assert(TAILQ\_NEXT(q,ev\_next)->val==1);
36. assert(p==TAILQ\_LAST(&queue\_head,queue\_head\_t)); //p为队列尾部元素
37. assert(TAILQ\_PREV(p,queue\_head\_t,ev\_next)->val==1);
38. t=new event\_t();
39. t->val=7;
40. TAILQ\_INSERT\_AFTER(&queue\_head,q,t,ev\_next);
41. cout<<"foreach"<<endl;
42. TAILQ\_FOREACH(s,&queue\_head,ev\_next){
43. cout<<s->val<<endl;
44. }
45. cout<<"foreach reverse"<<endl;
46. TAILQ\_FOREACH\_REVERSE(s,&queue\_head,queue\_head\_t,ev\_next){
47. cout<<s->val<<endl;
48. }
49. system("pause");
50. return 0;
51. }

        在event结构体中，则有几个TAILQ\_ENTRY(event)类型的成员变量。这是因为根据不同的条件，采用不同的队列把这些event结构体连在一起，放到一条队列中。